Machien Learning and Inductive Inference [H02C1a]

Xinhai Zou (r0727971)

Contents

1 Lecture 1: Introduction, Version spaces

1.1 Some ML examples in practice

1. Autonomous cars
2. The Robosail project
3. The Robot Scientist
4. Infra Watch, ”Hoolandse brug” - the bridge
5. Language learning
6. Automating manual tasks

1.2 Machine Learning

Definition of machine learning: it is the study of how to make programs improve their performance on certain tasks from own (experience).
In this case:

- ”performance” = speed, accuracy
- ”experience” = earlier observations

Machine Learning vs. other AI
In **machine learning**, the key is **data**, examples of questions and their answer; observations of earlier attempts to solve the problem
In **inductive inference**, it is reasoning from **specific** to **general**, statistics: sample -> population; from **concrete observations** -> **general theory**

1.3 Machine Learning learning landscape

- tasks
  - clustering
  - classification
  - regression
  - reinforcement learning
- techniques
  - Convex optimization
  - Matrix factorization
  - Transfer learning
  - Learning theory
  - Greedy search

- models
  - automata
  - neural network
  - deep learning
  - statistical relational learning
  - decision trees
  - support vector machines
  - nearest neighbors
  - rule learners
  - bayesian learning
  - probabilisitc graphical models
- applications
  - natural language processing
  - vision
  - speech
- related courses
  - neural computing
  - support vector machine
  - uncertainty in AI
  - data mining
  - genetic algorithms and evolutionary computing

1.4 Some basic concepts and terminology

- Predictive learning
  - Definition: learn a model that can predict a particular property/ attribute/ variable from inputs
  - Binary classification: distinguish instances of class C from other instances
  - Classification: assign a class C (from a given set of classes) to an instances
  - Regression: assign a numerical value to an instance
  - multi-label classification: assign a set of labels (from a given set) to an instance
  - multivariate regression: assign a vector of numbers to an instances

    - multi-target prediction: assign a vector of values (numerical, categorical) to an instances

- Descriptive learning
    - Definition: given a dataset, describe certain patterns in the dataset, or in the population it is drawn from

- Typical tasks in ML
    - function learning: learn a function X->Y taht fits the given data
    - distribution learning: distribution learning
        * parametric: the function family of the distribution is known, we only need to estimate its parameters
        * non-parametric: no specific function family assumed
        * generative: generate new instances by random sampleing from it
        * discriminative: conditional probability distribution

- Explainable AI (XAI)
    - Definition: means that the decisions of an AI system can be explained
    - Two different levels:
        * We understand the (learned) model
        * We understand the individual decision

1.5 Input formats (predictive learning)

- Set
    - training set: a set of examples, instance descriptions that include the target property (a.k.a. labeled instances)
    - prediction set: a set of instance descriptions that do not include the target property ('unlabeled' instances)
    - prediction task: predict the label of the unlabeled instances

- Outcome of learning process
    - transductive learning: the predictions themselves
    - inductive learning: a function that can predict the label of any unlabeled instance

- Explainable AI
    - interpretable: can be intepred
    - black-box: non-interpretable

- Learning

  - Supervised learning: from labeled
  - Unsupervised learning: from unlabeled
  - Semi-supervised learning: from a few labeled and many unlabeled

- Format of input data
  - input is often assumed to be a set of instances that are all described using the same variables (features, attributes)
  - i.i.d.: independent and identically distributed
    - tabular data (NN)
    - sequences
    - trees
    - graph
    - raw data: learning meaningful feaures from raw data
    - knowledge: inductive logic programming

1.6 Output formats, methods (predictive learning)

The **output** of a learning system is a model.

- output
  - parametrized functions
  - ocnjunctive concepts: a conjuntive concept is expressed as a set of conditions, all of which must be true
  - rule sets (if...then...else...)
  - decision trees
  - neural networks
  - probabilisitc graphical models
- search methods
  - discrete spaces - methods: hill-climbing, best-first
  - continuous spaces - methods: gradient descent
- typically
  - model structure not fixed in advanced - discrete
  - fixed model structure, tune numerical parameters - continuous
- hypothesis space
  - definition: all possible instances
  - for robot example: {B,R,M,?} x {S,T,?} x {L,W,?} x {1,2,?}

- Verson space
  - using candidate elimination
  - pros
    - can be used for discrete hypothesis spaces
    - search for all solutions, rather than just one, in an efficient manner
    - importance of generality ordering
  - cons
    - not robust to noise
    - only conjunctive concepts

2 Lecture 2: Induction of decision tree

2.1 Overview of DT

- A decision tree represents a decision procedure where
  - you start with one question
  - the answer will determine the next question
  - and repeat, untill you reach a decision
- We will usually call the questions ”tests” and the decision a ”prediction”
- attribute
  - input attribute X = {X_1, X_2 ..., X_n}
  - target attribute Y
  - the tree represents a function f: X -> Y
- Example: Playing Tennis Tree
  - Outlook: X_1 = {Sunny, Overcast, Rainy}
  - Humidity: X_2 = {High, Normal}
  - Wind: X_3 = {Strong, Weak}
  - Tennis: Y = {Yes, No}
  - The tree represents a function Outlook x Humidity x Wind -> Tennis
- Boolean tree
- Continuous input attributes
  - We cannot make a different child node for each possible value!
  - Solution: use comparative test -> a finite number of possible outcomes
- Type of trees
  - target attribute Y is nomial -> classification tree
  - target attribute Y is numerical -> regression tree
- **Advantages of Tree (Why tree?)**
  - Learning and using tree is **efficient**
  - Tend to have **good predictive accuracy**
  - Tree is **interpretable**

2.2 Learn trees from data

- Two tasks for DT
  - Task 1: find the smallest tree T such that ∀(x,f(x))∈D: T(x)=f(x) (meaning that only fullfill current data set)
  - Task 2: find the tree T such that for x drawn from population *D*, T(x) is (on average) maximally similar to f(x) (T:model tree from data set D, f(x):true function in population *D*)
    - loss function: l: Y_1 x Y_2 -> R (where Y_1 is predicted value, Y_2 is actual value)
    - risk R of T, the expectation of loss function, is $E_{x \sim D}[l(T(x), f(x))]$, which is needed to be minimal.
- the basic principle
  - The approach is known as ”Top-down induction of decision trees (TDIDT)”, or ”recursive partitioning”
    - 1. start with the full data set D
    - 2. find a test such that examples in D with the same outcome for the test tend to have the same value of Y
    - 3. split D into subsets, one for each outcome of that test
    - 4. repeat this procedure on each subset that is not yet sufficiently ”pure” (meaning, not all elements have the same Y)
    - 5. keep repeating until no further splits possible
- rule representation of tree
  - trees can be written as if-then-else rules
  - rules can be simplified
- Two main questions?
  - How to choose which test should be the first? (guess: attribute with minimal entropy?)
  - When to stop splitting nodes? (guess: till pure? or threshold for probability?)

2.3 Choosing the best test

- We focus on classification tree (Y is nominal)
- Information theory
  - a good test is a test that carries much information about the class
  - ”entropy” or ”missing information”: how many bits needed, on average, to convey a piece of infomration, if an optimal encoding is used

- **bits** <- Question!! Do not understand the bit, how is it related to entropy?
- But whatever, the cleverest bit has been provided as below, which is called "entropy"
  - $e = -\sum_{i=1}^{k} p_i \log_2 p_i$
- The number e reflects the minimal number of bits that you will need, on average, to encode one value. It is the inherent information content, or **entropy**.

- for classification - we use class entropy
  - The class entropy of a set S of objects($\mathbf{x}$,y), where y can be any of k classes c$_i$, is defined as
    - $CE(S) = -\sum_{i=1}^{k} p_i \log_2 p_i$ with $p_i = \frac{(|(x,y) \in S | y = c_i|)}{|S|}$
    - it measures how much uncertainty there is about the class of a particular instance
  - high entropy = "many possbilities, all equally likely" - not good for splitting
  - low entropy = "few possibilities, safer to conclude" - better for splitting nodes
  - entropy measures uncertainty
- information gain (IG)
  - will have the same effect of attribute entropy
  - the information gain of a question = the expected reduction of entropy by obtaining the answer to the question
  - in the case of classification trees: expected reduction of class entropy:
    - $IG(S,t) = CE(S) - \mathbb{E}(CE(S_i)) = CE(S) - \sum_{i=1}^{o} \frac{|S_i|}{|S|} CE(S_i)$
    - with t a test, o the number of possible outcomes of attribute/test t, and S_i the subset of S for which the i'th outcome was obtained.

However, if we focus on regression tree (Y is numerical), can we still use class entropy or information gain?

- Now we assume Y is numerical
- Now we use **variance reduction** instead of entropy or information gain
  - the variacne of Y in a set S of instances (x,y) is
  - $Var(S) = \frac{\sum_{(x,y) \in S} (y - \hat{y})^2}{|S| - 1}$ with $\hat{y} = \frac{\sum_{(x,y) \in S} y}{|S|}$
  - and the variance reduction will be (which is similar to information gain (IG))
  - $VR(S,t) = Var(S) - \sum_{i=1}^{o} \frac{|S_i|}{|S|} Var(S_i)$

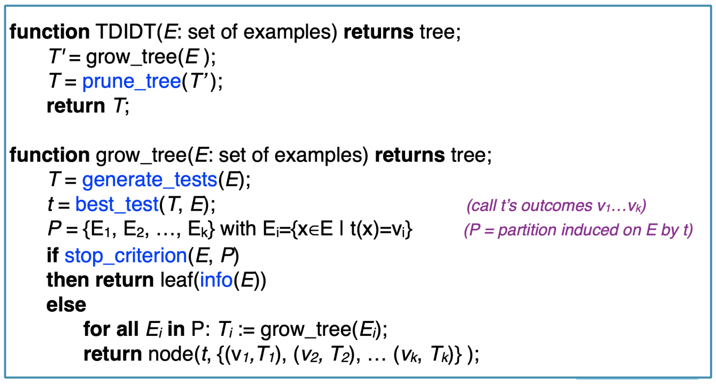
2.4 Stop splitting nodes

- In principle, keep splittign untill all instances in a subset have the same Y value
  - However, this is useful for task 1, but less useful for taks 2 (may cause overfitting problem!)
  - Please remember this is not population, this is just a sampling sample.
- overfitting
  - overfitting improves the consistency of the tree with the given data set D, but may decrease accuracy for instances outside D (whole population $\mathcal{D}$)
- How to avoid overfitting?
  - ”cautious splitting”: do not split a node unless you are certain that the split is meaningful
  - ”post-pruning”: do not bother about overfitting while splitting nodes, but once the (large) tree has been built, prune away branches that turned out not to contribute much
- ”Cautious splitting”
  - How do we know when not to split a node any further?
    - a simple approach: try to guess when accuracy on unseen data is going down (**”the turning point”**)
  - But how canwe guess this turning point, if the data is unseen?
    - Solution: we can use ”validation data” to evaluate
    - this ”validation data” is not for growing tree, only for estimating accuracy on ”unseen” data
- ”Post-pruning”
  - What if the accuracy will increase again? - afraid to end the growth too early
    - Solution: we can compute the whole data set, then find its highest point
  - Princeple
    - grow the tree to its full size, then cut away branches that did not contribute to getting better predictions
    - How to decide which branch does not contribute
      - check for each node in the tree, starting at the bottom: ”what would be the accuracy if I cut the tree here?” - if that accuracy is not lower, cut the tree, otherwise, do not cut.

- Pros and Cons
  - post-pruning requires more effort to build a large tree, while it gives more accurate tree
  - catious splitting is more efficient, while the accuracy is not as good as post-pruning

2.5 A generic algorithm

- TDIDT = "Top-down induction of decision trees", also referred to as "recursive partitioning"
- most decision tree learners follow the same basic approach, but differ in the details
  - we will have a look at the commonalities and differences



```
function TDIDT(E: set of examples) returns tree;
    T' = grow_tree(E );
    T = prune_tree(T');
    return T;

function grow_tree(E: set of examples) returns tree;
    T = generate_tests(E);
    t = best_test(T, E);                          (call t's outcomes $v_1 \ldots v_k$)
    P = {$E_1$, $E_2$, …, $E_k$} with $E_i$={x∈E | t(x)=$v_i$}    (P = partition induced on E by t)
    if stop_criterion(E, P)
    then return leaf(info(E))
    else
        for all $E_i$ in P: $T_i$ := grow_tree($E_i$);
        return node(t, {($v_1$,$T_1$), ($v_2$, $T_2$), … ($v_k$, $T_k$)} );
```

Figure 1: Algorithm for "Top-down in duction of decision trees"

- The blue functions are where implementations differ
  - prune_tree: how to prune the tree afterward
  - generate_tests: which tests to consider
  - best_test: which test to select (use heuristics: entropy or variance)
  - stop_criterion: when to stop (cautious splitting or post-pruning)
  - info: what information to store in the leaf
- generate_tests
  - for numerical attributes: oblique trees can be used for determining c(threshold), while it will be more difficult even though it has higher accuracy. Thus, in practice, non-oblique trees are much more common.

- best_test
  - for classification trees: information gain (IG) = reduction to entropy $CE(S) = -\sum_{i=1}^{k} p_i \log_2 p_i$
    - in some cases: "Gini impurity" instead of entropy: $Gini(S) = 1 - \sum_{i=1}^{k} p_i^2$
  - for regression trees: reduction of variance σ, or reduction of standard deviation $\sqrt{\sigma}$
    - in some cases: normalization is impelmented by "Split information" (SI): $SI(S,t) = -\sum_{i=1}^{n} \frac{|S_i|}{|S|} \log_2 \frac{|S_i|}{|S|}$
    - and the Gain Ratio (GR) will be: $GR(S,t) = \frac{IG(S,t)}{SI(S,t)}$
  - for numerical inputs, how to determine c?
    - Solution: typically, all values are tried, and the c that yields the best heuristic value (IG, GR, Gini, ...) is selected (best of them)
- info
  - for classification trees: usually, the most frequent class
  - for regression trees: usually, the mean of all target values in that leaf
    - possible: median
- stop_criterion
  - 1. cautious splitting, post-pruning
  - 2. threshold
    - classification: all instances have the same class, pure
    - regression: variance
  - too few examples to continue splitting
    - only allow tests that yield subtrees with at least two examples each
- impurity measures
  - good impurity measures are strictly concave <- Question: do not understand!!

2.6 Computational complexity

- Given a data set D consisting of N instances (x,y), where x has m components (attribtues), how do N and m influence the computational effort required to learn a tree?
- "Splitting one node" - for each node, we need to "find the best test"
  - for each possible test, evaluate its quality

    * partition the dataset according to this test
    * compute quality based on this partition
  - for the test with highest quality
    * partition data according to that test

- Efficiently computing test quality for continous attribtues
  - first **sort** all tuples according to A (attribute), small to large
  - then gradually move the threshold, and simply update the tables
  - thus, testing **all thresholds** is only slightly more work than testing one!
- Computing the quality of one test
  - for nominal attributes: 1 scan, gradually building the table; compute IG once, at the end
  - for continous attributes: 1 scan, gradually updating the table; compute IG for each intermediate table
  - hence, computing the best test for a node is linear in the number of instances in that node $O(n)$, with n the numebr of instances
  - Complexity:
    * compute quality of one test is $O(n)$
    * compute quality of all tests is $O(nm)$, with m the number of attributes
- Splitting multiple nodes
  - the total at one level of the tree is always n, so the overal amount of work depends on the layer of the trees
    * if the splits are balanced, the height of tree is $O(log(N))$, and the overall compelxity of tree growth is $O(mN\log_2(N))$
    * if the splits however are unbalanced, the height of tree is $O(N)$, and the complexity is $O(mN^2)$
- For impurity measures
  - ”pure” = ”zero variance/entropy”
- Why decision tree for Big Data?
  - QUICK! and FAST! efficient!
  - high accuracy
  - interpretable can be also a reason

2.7 Missing values

- when computing quality of a test: just ignore instances where this value is missing
- when splitting on an attribute:
  - guess its most likely value
  - partially assign the example to multiple branch, with its probability

2.8 Model trees

Model trees are **regression tree** in which each leaf does not contain a constant, but a linear model.

- RETIS (M5)
- Mauve

2.9 Multi-target trees

- Classification and regression trees are very popular for predicting one target variable
- But the principle of variance reduction is easily generalized to predicting vectors!
- This allow us to predict multiple variables at the same time, using one tree. **(vector)**
- Multi-label classification
  - How? For example for Y = {a,b,c,d,e}
  - Option 1: ”binary relevance”
    - ∗ learn (y/n) decision tree for each label
    - ∗ TreeA predicts a(y/n), TreeB predicts b(y/n)
  - Option 2: ”label powersets”
    - ∗ consider each set as separate label, for 5 original labels a,b,c,d,e, we get 32 combined labels: -,a,b,c,d,e,ab,ac, ..., abc, ...
    - ∗ Learn a tree that predicts the combined label
  - Option 3: ”Vector encoding”
    - ∗ encode each set as 0/1 vector, e.g. {a,b,d} = [1,1,0,1,0]
    - ∗ use a learner that can learn models that predict vectors
  - Option 3: ”Vector encoding” is better! Requires almost no changes in the algorithm.

    * for example a leaf contains {[1,1,0,1,0],[1,1,0,0,0],[1,1,0,1,1]} -> [1,1,0,0.67,0.33]
    * predict all labels by threshold (e.g. 0.5): [1,1,0,0.67,0.33] -> {a,b,d}

- Hierarchical multilabel classification (HMC)
  - for protein
  - similar just add a constrant ”a label can only occur in an instance’s label set if all its ancestors also occur”
    * 250 functions = 250 trees is not fast and interpretable!
    * 250 functions = 1 tree is fast and interpretable!
    * learn 1 tree that predicts 250-dimensional class vector
  - Decision tree can be converted to rules set

2.10 Practical software

- Weka
- PythonL scikit-learn
- others: R, SAS, SPSS

3 Lecture 3: Learning sets of rules

3.1 Linear regression

Decision tree vs. linear regression

- "Linear model": $Y = a + b_1X_1 + b_2X_2 + ... + b_kX_k$
  - usually fit such that sum of squared vertical deviations from line is minimal
  - what can we learn from such linear model?
    - for predicting Y, given Xi
    - for understanding how well Y can be predicted from Xi
    - for understanding what the effect of each Xi is on Y
    - for visualizing the connection between Xi and Y
    - the linear correlation r tells us how well the points fit a line $-1 \leq r \leq 1$
    - the coefficient of determination R^2 tells us to what extent Y is determined by the X_i, $0 \leq R^2 \leq 1$
  - careful with interpretation of coefficients
    - coefficients are not scale-free
    - "multicollinearity": correlations among X_i
- important assumptions
  - effect of each variable on target is constant (does not depend on other variables)
  - effect of different variables are cumulative (add up)
- complex terms
  - "overall" coefficient of X_2 is $(b_2 + b_{12}X_1)$ effect of X_2 on Y depends on X_1
- nominal variables
  - for nominal X_i with k values, introduce k-1 "0/1" variables, called "indicator" or "dummy" variables <- Question: do not understand the dummy!!!

3.2 Trees vs. linear regression vs. inductive bias

- Each learning approach has a "bias": implicit assumptions it makes.
- Removing all bias?
  - bias-free learning is imporssible!

    - no single best method for learning!
    - for VS, the bias is ”it assumes conjunctive concepts”. -> without bias, no generalization.
    - all of learning models have their own bias = implicit assumptions about what properties the true model has

- Choices to make
    - Modelling your problem as a prediction tasks:
        - What is input, what is output (target attribute)?
        - Regression, classification, probability prediction?
    - Choosing a learning approach
        - Efficiency of learning/prediction phase
        - Bias (which more fits the problem)
        - interpretability of returned models (interpretation)

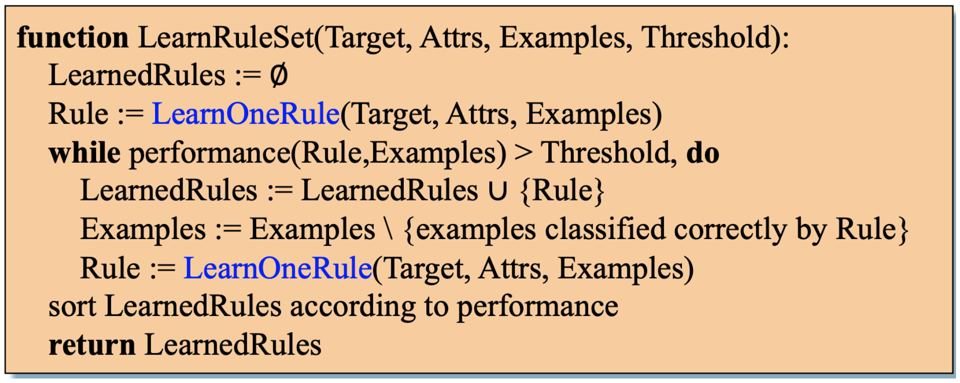
3.3 Rule learning

Learning sets of classification rules: rule sets:

1. ”if...then...”
2. ”if...then...else...”

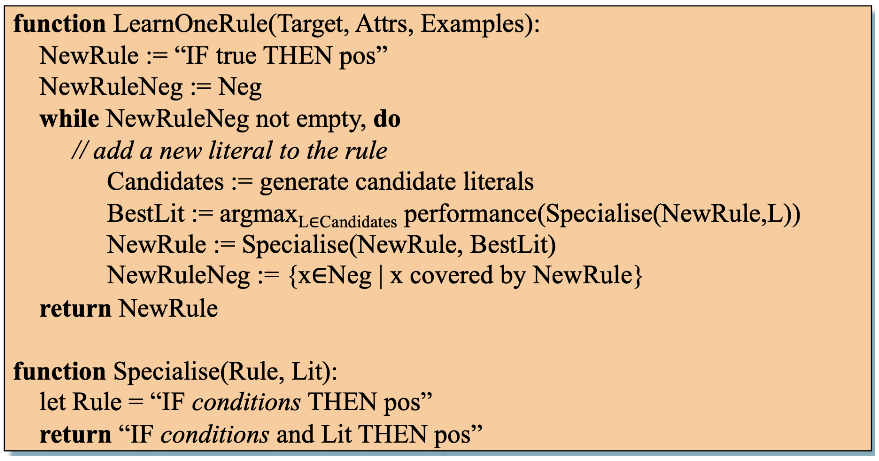
- Example: rule sets - define a leap years
    - If year is multiple of 400 then leap
    - else if year is a multiple of 100 then not leap
    - else if year is multiple of 4 then leap
    - else not leap
- A decision tree can be turned into a set of rules!
    - By learning decision tree to learn rule sets
- principle
    - 1. High accuracy: when it makes a prediction, it should be correct
    - 2. Reasonable coverage: it needs not make a prediction for each instance, but the more, the better
- Coule be top-down or bottom-up
    - Top-down:
        - Start with maximally generally rule
        - add literals one by one

        * gradually maximize accuracy without sacrificing coverage
    - Bottom-up:
        * Start with maximally specific rule
        * remove literals one by one
        * gradually maximize coverage without sacrificing accuracy



```
function LearnRuleSet(Target, Attrs, Examples, Threshold):
    LearnedRules := ∅
    Rule := LearnOneRule(Target, Attrs, Examples)
    while performance(Rule,Examples) > Threshold, do
        LearnedRules := LearnedRules ∪ {Rule}
        Examples := Examples \ {examples classified correctly by Rule}
        Rule := LearnOneRule(Target, Attrs, Examples)
    sort LearnedRules according to performance
    return LearnedRules
```

Figure 2: Algorithm for "General algorithm for rule learning"



```
function LearnOneRule(Target, Attrs, Examples):
    NewRule := "IF true THEN pos"
    NewRuleNeg := Neg
    while NewRuleNeg not empty, do
        // add a new literal to the rule
            Candidates := generate candidate literals
            BestLit := argmax_{L∈Candidates} performance(Specialise(NewRule,L))
            NewRule := Specialise(NewRule, BestLit)
            NewRuleNeg := {x∈Neg | x covered by NewRule}
    return NewRule

function Specialise(Rule, Lit):
    let Rule = "IF conditions THEN pos"
    return "IF conditions and Lit THEN pos"
```

Figure 3: Algorithm for "General algorithm for learning one rule"

- Top-down: start with an empty rule
- Heuristics for rule learners
    - High accuracy (most important), it is not robust to noise

  - reasonably high coverage
- if-then-else rules vs. decision lists
  - if-then-else rules
    - if year is a multiple of 400 then leap
    - else if year is a multiple of 100 then not leap
    - else if year is multiple of 4 then leap
    - else not leap
  - decision lists
    - if year is a multiple of 400 then leap
    - if year is multiple of 4 but not of 100 then leap
  - if-then-else rule is more interpretable
  - decision list is more compact
  - unordered rules vs. ordered rules <- Question: do not understand!!!
- example-driven top-down rule induction
  - works like regular top-down approach, except:
    - pick a not-yet-covered example
    - consider as hypothesis space, all the rules that cover this example
    - search within this hypothesis spaces (much smaller)
  - **pros:** more efficient
  - **cons:** less robust to noise
- other examples: RIPPER, Weka (software)

3.4 Association rules

Table 1: Differences with Classification and association rules

Classification rules	**Association rules**
One target class	Any combinatino of items can be the target
Good rules have near 100% accuracy ("confidence" here)	Rules need not have near 100% confidence to be interesting
Find a minimal set of rules (just enough to classify)	Find a maximal set of rules (all rules that hold)

- Overview
  - Similar to classification rules, but for descriptive learning instead of predicitve learning
  - is to look for the patterns in data

  - classification rules are a small subset of association rules?
- General format: if $a_1, a_2, ..., a_n$ then $a_{n+1}, a_{n+2}, ..., a_{n+m}$
- Rule "If <this> then <that>" is charaterized by
  - Support: % of all clients that buy <this>
  - Confidence: % of buyers of <this> that also buy <that>
- Running a classification rule learner clearly will not work well for association rule, since classification rule is only a small subset of association rule
  - Solution: APRIORI algorithm

4 Lecture 4: Instance-based learning, Clustering

4.1 Instance-based learning

Basic key idea: just store all training examples

- When seeing a new instance:
  - Find the most similar cases in the database
  - make a prediction based on those instance
- Architypical method: k-nearest-neighbors (k-NN)
  - use most frequent class/ mean target among the k nearest neighbors as your prediction
  - k is chosen by the user (hyperparameter)
- Similarity
  - how close to others, often Euclidean distance for numerical inputs
- Voronoi diagrams
  - indicate area where prediction is influenced by same set of examples
  - for 1-NN: cell borders are right in the middle between any two data points
  - This is called a Voronoi diagram - kNN
- Decision surface
  - Decision surface separate regions with different predictions
- Voronoi diagram for k > 1
  - To construct diagram for k-NN with k > 1
    - Start from diagram for k-1
    - for each cell:
      - temporarily forget about k-1 nearest neighbors
      - split cell according to k'th nearest neighbors
    - Merge adjacent cells with same k nearest neighbors
- pros vs. cons
  - **pros**
    - "Learning" is very fast - just storing the data
    - all detials of the data are kept
  - **cons**

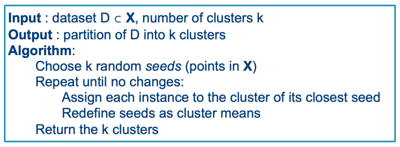
        - can be slow at prediction time
        - difficulties in high-dimensional spaces
        - relies on having a good similarity measure (Euclidean distance - numerical)
        - not robust to noise

- for k-NN, large K -> more robust to overfitting? but it is more robust to noise
- difficulties with high-dimensional space - curse of dimensionality - data are distribtued very sparsely in high-D
- Improvement: Different scales
  - When dimensions have very differnet scales, Euclidean distance may not work well
    - Solution: normalization - normalize all dimensions to comparable scale
    - Give irrelevant dimensions a smalelr weight in the Euclidean distance, so they have less influence
    - using a closed formula: $w_i = 1 - \frac{1}{n} \sum_{k=1}^{c} \sum_{j=1}^{n_k} |\bar{x}_{ki} - x_{kji}|$
    - with c=#clusters, n_k=#elements in cluster k, x_{ki}=mean x_i in cluster k, x_{kji}=x_i for jth element in cluster k
- Improvenment: distance-werights k-NN
  - Why 3-NN, but why not 4-NN
    - Solution: no cut-off at k, but have weights gradually decrease with distance
    - careful: influence must decrease fast with distance, otherwise faraway cases will dominate the voting (otherwise noise)
- Improvement: locally weighted regression
  - for better fitting
    - Solution: fit a simple local model
    - a linear regression can be okay
- Prototypes
  - A prototype is a representattive for a group of instances
  - can be an average for elements in a cluster
  - note: need to define similarity between instance & prototype
- Efficient prediction
  - Solution: indexing the data helps

- Lazy learners vs. eager learners
  - k-NN is called lazy learner
    - do not build a model during training, merely store data
    - start doing the hard work when asked a question
  - eager learners
    - generalize before knowing the question
    - try to build a global model that will owrk under all circumstances
  - lazy learners can use simpler local models - sometime very accurate
    - compare: locally weighted linear regression (k-NN) vs. global linear regression
    - compare: learn one rule that covers the query instance vs. learn a global rule set

4.2 Clustering

- Overview
  - Find structure/pattern in the data, in the form of groups of instances that are highly similar to each other
  - It is unsupervised machine learning (no labeled samples)
- Find groups (clusters) of instances so that
  - instances in the same group are similar
  - instances in the difference group are different
- Definition of clustering
  - flat clustering
    - returns a partition of the data
  - hierarchical clustering
    - returns a hierarchy of clusters
- Other definition of clustering
  - extensional clustering
    - clusters are defined without any description language
  - conceptual clustering
    - clusters are defiend using a conceptual description language
- FLat, extensional clustering



```
Input : dataset D ⊂ X, number of clusters k
Output : partition of D into k clusters
Algorithm:
    Choose k random seeds (points in X)
    Repeat until no changes:
        Assign each instance to the cluster of its closest seed
        Redefine seeds as cluster means
    Return the k clusters
```

Figure 4: Algorithm for ”Algorithm for flat, extensional clustering: K-means”

  - task: given a set of unlabeled data, find groups of highly similar instances
  - some constraints may additionally be given
  - procedure: reassign points, recompute seeds, reassign points ..., when no points are re-assigned to another cluster, stop.

- Hierarchical, extensional clustering
  - top-down (”divisive”) methods
    - start with 1 cluster (whole data set)
    - divide it into subsets
    - subdivide subsets further
  - bottom-up (”agglomerative”) methods
    - start with singleton clusters
    - join closest clusters together
    - repeat until 1 cluster
- Bottom-up methods
  - How does it generalize to distance between clusters? - based on examples
    - Single linkage: distance between clusters = distance between each closest point in their cluster
    - Complete linkage: distance between clusters = distance between each furthest poin in their clusters
    - Average linkage: distance between clusters = distance between average of clusters
- Conceptual clustering
  - Cluster + find a conceptual description of each cluster (in some given description language L)

  - Clearly, the clusters are not defined using distance alone! Context is improtant!

- Examples: Cobweb
  - Predictability: given cluster, how well can you predict attribute values
  - Predictiveness: given attribute values, how well can you predict cluster
  - Cobweb is to maximize a combination of both

- Decision tree learning, viewed as clustering
  - A decision tree defines a conceptual, hierarchical clustering of the data
    * each node = subset of the data
    * conceptual description of these data = conjunction of test outcomes from root to node

- Clustering trees
  - for regression (original), heuristic minimize average variance of Y within subsets
    * $Var(S) = \sum_{(x,y)\in S} \frac{(y-\bar{y})^2}{|S|-1|}$
    * with $\bar{y} = \sum_{(x,y)\in S} \frac{y}{|S|}$
  - for clustering (unsupervised clustering tree), given some distance metric d that indicates dissimilarity, we can minimize the average variance of X wihtin subsets
    * $Var(S) = \sum_{x\in S} \frac{d(x,\bar{x})^2}{|S|-1}$
    * with $\bar{x} = \sum_{x\in S} \frac{x}{|S|}$

- Predictive clustering
  - overview: it looks like from value -> cluster and from cluster -> value
  - Predictive clustering builds a model consisting of
    * a set of clusters
    * a function c assigning instances to clusters
    * a function p assigning a target value to the instance, given the cluster
  - the overall predictive function is then $f(x) = p(c(x), x)$
  - the accuracy of f will depend on that of c and p.
    * an accurate c requires good predictiveness (from value to cluster)
    * an accurate p requires good Predictability (from cluster to value)

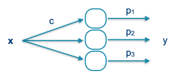


Figure 5: Algorithm for ”Scheme of Predicttive Cluster”

- Similarity measures
  - Similarity is often represented using a distance metric
    - small distance = high similarity, e.g., euclidean distance, manhattan distance, hamming distance, ...
    - $d_{Eucl}(x, x') = \sqrt{\sum_i (x_i - x'_i)^2}$
    - $d_{Manh}(x, x') = \sum_i |x_i - x'_i|$
  - Not all similarity measures can be expressed in that way!
    - Distance metric fulfills symmetry and triangle inequality
    - Some similarity measures cannot be mapped to such a distance measures
- Learning the similarity measure
  - Clustering relies strongly on defining an appropriate similarity measure
  - However, this may be very difficult sometime
    - Solution: semi-supervised clustering
    - Computer learns the similarity from examples of pairs of instances that should be in the same/ in differrent clusters, ”must-link” and ”cannot-link” constraints
    - then start the unlabeled data, (semi-supervised clustering: a few labeled data, many unlabeled data)
- Learning preferred clusterings
  - Different clustering algorithms have different biases
    - k-means: spherical clusters
    - density-based: can return concave clusters
    - some methods can even return disconnected cluster
- example of interactive clustering: the COBRAS method

  - using an intermediate level of super-instances
    * clustering = set of clusters
    * cluster = set of super-instances
    * super-instance = set of clusters

- Evaluating clusterings
  - How can we assess whether a clustering is good or bad?
  - explicit objective: minimize intra-cluster variance
  - internal criteria: inherent to the clustering
  - external criteria: compare to a reference clustering
    * rand (random) index
    * adjusted rand (random) index (ARI)

5 Lecture 5: Evaluating hypotheses

5.1 Models and learning algorithms

- Different levels of evaluation
  - evaluation of learned models
    - given the model, how well can we expect it to perform
  - evaluation of learning algorithm
    - given an algorithm, how well can we expect the models it learns to perform
- Evaluation of classifiers
  - probability of making a correct prediction: accuracy
  - time needed to make the prediction
  - cost incurred by wrong predictions

5.2 Statistics

- accuracy
  - accuracy = probability of correct prediction on a randomly drawn instances
  - error = 1- accuracy
  - estimating accuracy = estimating a probability over a population
  - estimating probability from sample proportion: well-studied problem in inferential Statistics
  - for probability π, sample proportion p has $\mathbb{E}(p) = \pi$ and $Var(p) = \frac{\pi(1-\pi)}{n}$
  - Hence:
    - point estimate: $\hat{\pi} = p$
    - 1-α confidence interval: $[p - z_{\alpha/2}\sqrt{\frac{p(1-p)}{n}}, p + z_{\alpha/2}\sqrt{\frac{p(1-p)}{n}}]$ with $z_{\alpha/2}$ such that $P(Z > z_{\alpha/2}) = \alpha/2$ if $Z \sim N(0,1)$

Table 2: Alpha for Confidence interval

α	**0.1**	**0.05**	**0.01**
$z_{\alpha/2}$	1.64	1.96	2.56

- train set and cross-validation
- option 1: train/test split

- use 2/3 of available data for training set T, set aside 1/3 for test set S
- while if T is smaller -> less accurate f, smaller S -> less accurate estimate of accuracy of f

- option 2: cross-validation
  - learn f from the full data set S
  - however to get an estimate of acc(f), we parition the set into n subsets S_i
  - learn f_i, i=1...n from $\frac{S}{S_i}$, compute $ACC(f, S_i)$
  - estiamte acc(f) as average of all $acc(f, S_i)$
  - name
    - for a specific n, this is called n-fold cross-validation
    - when n = |S|, this is called leave-one-out cross-validation
  - Is cross-validation entirely unbiased - No
  - More option: nested/internal cross-validation

- comparing two models
  - given two models, which one has higher accuracy
  - two cases
    - compare 2 models on different test sets
    - compare 2 models on same test sets
  - different sets
    - Checked by confidence interval: $(a_1-a_2) \pm z_{\alpha/2}\sqrt{\frac{a_1(1-a_1)}{n_1} + \frac{a_2(1-a_2)}{n_2}}$
    - if CI for a_1-a_2 is entirely to the right of 0, meaning that $a_1-a_2 \geq 0$, meaning that $acc(f_1) \geq acc(f_2)$
    - if CI for a_1-a_2 is entirely to the left of 0, meaning that $a_1-a_2 \leq 0$, meaning that $acc(f_1) \leq acc(f_2)$
    - if CI contains 0, it means that there is no difference between f_1 and f_2
  - same sets
    - Compare models on the same data set is more informative
    - uses more detailed information from test
    - use McNemarś test in this case
    - key idea: how often was f_1 right and f_2 wrong on the same example vs. how often was f_1 wrong and f_2 right on the same example
      - if $B \approx C \approx (B + C)/2$, then accept $acc(f_1) = acc(f_2)$

      · otherwises, if B deviates too much from $(B+C)/2$, then reject $acc(f_1) = acc(f_2)$
    * McNemarś test is more informative - but can only use it when individual predictions of both models on the same test set are available

Table 3: Table for McNemarś test

	f_1 **right**	f_1 **wrong**
f_2 **right**	A	B
f_2 **wrong**	C	D

- Evaluation based on accuracy is not always appropriate
- cons
  - no obvious reference point
    - sample imbalance
  - unstable when class distribution may change
  - *assumes symmetric misclassification costs*
- alternatives
  - correlation
  - ROC analysis
- correlation
  - correlation tends to be more informative for unbalanced classifiers
  - $\phi = \frac{AD-BD}{\sqrt{T_+ \cdot T_{T_-} \cdot T_{pos} \cdot T_{neg}}}$ with 1: perfect prediction, 0: no correlation (totally random), -1: predicting the exact opposite (1 and -1 both indicate strong relation, more focus on absolute number)

Table 4: Real class vs. predicted class

	+	-	**Total**
pos	A	B	T_{pos}
neg	C	D	T_{neg}
Total	T_+	T_-	T

- expected misclassification cost
  - sometimes one type of mistake is worse than the other, accuracy ignores this (assumes all errors are euqally bad -> which are not correct usually)

  - Solution
    * accuracy: probability that some instance is classified correctly
    * TP: ”true positive rate”: probability of a positive to be classified correctly
    * TN: ”true negative rate”: probability of a negative to be classified correctly
    * FP: ”false positive rate”: $FP = 1 - TN$.
    * FN: ”false negative rate”: $FN = 1 - TP$.
  - consider ”misclassification costs”
    * C_{FP}: cost of a false positive (cost of classifying a - as pos)
    * C_{FN}: cost of a false negative (cost of classifying a + as neg)

6 Lecture 6: Numerical approaches (ANN, SVM), Computational learning theory

7 Lecture 7: Probabilistic approaches, Ensembles

8 Lecture 8: Reinforcement learning

9 Lecture 9-10: Inductive logic programming